Lebanese University
Faculty of Science
BS Computer Science
2nd Year – S3

I2204 - Imperative Programming

Dr Siba Haidar

Lebanese University
Faculty of Science
BS Computer Science
2nd Year – S3

Linked Lists Exercises

Chapter 4

Linked List Exercises

1. Linked List of Contacts
2. Linked List of Integers
3. Linked List of Students
4. Linked List of Bank Accounts
5. Polynomials: Linked List of Terms
6. Doubly Linked List
7. AMSTRAMGRAM: Circular Linked List

Exercise: type contact

- contact type is defined to manage **sorted** linked list of contacts
- data stored concerns
  - name
  - tel (int)
- define the data type contact

Exercise: Push contact

- write the **Push** function which pushes a new node to the head of a contact list
  - (push does not care of the order)
  - the function type must be void

Exercise: printContact

- write the function printContact which prints a given contact

Exercise: printContacts

- write the function printContacts() that prints out the content of a given contact list, a contact per line
  - iterative version
  - recursive version

Exercise: deleteContact

- write the function deleteContact() that given a list of contacts, and a name, deletes the contact holding that name from the list
  - only first iteration
  - all of them

  - using localRef or not
  - iterative version
  - recursive version

Exercise: deleteList

- write the function deleteList() that takes a list of contacts, deallocates all of its memory and sets its head pointer to NULL (the empty list)

Exercise: insert

- write the function **insert()** that inserts a given name and tel number in a given list
  - in the right place → remember they are sorted
  - iterative version
  - recursive version
- hint : think of using the Push function, with the appropriate parameters

Exercise: numberOf

- write a function numberOf() that returns the telephone number of a given name
  - of the first occurrence

Exercise: displayAll

- write a **recursive** function displayAll() that displays all the contacts having a given name

Exercise: occurrence

- write the function occurrence() that returns the number of times a given name occurs in a given list
  - iterative version
  - recursive version

Linked List Exercises

1. Linked List of Contacts
2. Linked List of Integers
3. Linked List of Students
4. Linked List of Bank Accounts
5. Polynomials: Linked List of Terms
6. Doubly Linked List
7. AMSTRAMGRAM: Circular Linked List

Important

- In the following, we will use the following declaration:

```
typedef struct node{
  int data;
  struct node* next;
} node;
```

Exercise: count

```
typedef struct node{int data; struct node* next;} node;
```

- write the function count (iterative & recursive)
  - returns the number of nodes in a list

```
void countTest(){
      node* head=buildList();
      printf("%d", count(head));          //7
}
```

Exercise: printList

```
typedef struct node{int data; struct node* next;} node;
```

- write the function printList (iterative & recursive)
  - prints on the screen the values in data field of nodes

- example

```
void printListTest(){
    node* head=buildList();
    printList(head);          // 1,3,5,6,3,8,9
}
```

Exercise: printRList

- Write a function named printRList that prints the list elements in reverse order.

```
void printRList Test(){
      node* head=buildList();
      printRList(head);       // 9,8,3,6,5,3,1
}
```

Exercise: averageList

- Write a function named averageList that prints the average of nodes in the list.

```
void averageListTest(){
      node* head=buildList();
       printf("%lf",averageList(head));   // 5.0
}
```

Exercise: isSortedList

- Write a function named isSortedList that checks whether the nodes in a list are sorted in ascending order.

```
void isSortedListTest(){
      node* head=buildList(),*head2=buildListSorted();
      printf("%d",isSortedList(head));    // 0
      printf("%d",isSortedList(head2));   // 1
}
```

Exercise: incList

- Write a function named incList that adds a number passed as a parameter to each element of a given list.

```
void incListTest(){
      node* head=buildList();
      addNumberToList(head,2);
      printList(head);        // 3,5,7,8,5,10,11
}
```

Exercise: isRepeatedInList

- Write a function named “isRepeatedInList ” that checks whether a number is repeated in the list.

```
void isRepeatedInListTest(){
	node* head=buildList();
	printf("%d", isRepeatedInList(head,3)); // 1
	printf("%d", isRepeatedInList(head,8)); // 0
	printf("%d", isRepeatedInList(head,4)); // 0
}
```

Exercise: swapInList

- Write a function named “swapInList ” that swaps the contents of 2 nodes (of index `i` and `j`) in the list

```
void swapInListTest(){
	node* head = buildList();
	 swapInList(&head,1,3);	// swaping 3 by 6
	printList(head);	// 1,6,5,3,3,8,9
}
```

Exercise: removeFromList

- Write the function removeFromList which removes from a given list, the nodes having the given value.
  - iterative
  - recursive

```
void removeFromListTest(){
  node* head=build123();
  removeFromList(&head,1);
  printList(head);
 }
```

Exercise: insertNthList

- Write the function insertNthList which inserts in a given list, a given value, at a given index.
  - iterative
  - recursive

```
void insertNthListTest(){
  node* head=build123();
  insertNthList(&head,1, 2);
  printList(head);
 }
```

Exercise: PushToEnd

- Write the function insertNthList which inserts in a given list, a given value, at a given index.
  - iterative
  - recursive

```
void PushToEndTest(){
  node* head=build123();
  PushToEnd(&head,4);
  printList(head);
 }
```

List Application

- Write a complete Linked List Application with a menu.
- in a while(1) loop calls
  - menu() → see next slide
  - scanf()
    - if 0 exit from loop
    - if nb between 1 and 8
      - call the “call” function → see next slides

- function to print menu:
  - Enter your choice,
  - 1- create a empty list of integers
  - 2- display your list
  - 3- add one node to the head of your list
  - 4- add one node to the tail of your list
  - 5- delete one node containing the value v
  - 6- delete all the nodes containing the value v
  - 7- free the list
  - 8- insert a value at given index (insertNth)
  - Etc

Example to a call function

- suppose the user enters the choice 8
- insert a value at given index
- the function insertCall() will be called:
- void insertCall(node ** headRef){

```
    int value, index;
    //read v from keyboard..
   printf("please enter the value and the index");
   scanf ("%d %d", &value, &index);
   //call add to insertNth
   insertNth (headRef, index, value);
}
```

Linked List Exercises

1. Linked List of Contacts
2. Linked List of Integers
3. Linked List of Students
4. Linked List of Bank Accounts
5. Polynomials: Linked List of Terms
6. Doubly Linked List
7. AMSTRAMGRAM: Circular Linked List

Exercise: major

```
typedef struct Student{char name[20]; int id, grades[6];}std;
typedef struct node{std data; struct node* next;}node;
```

- write a function major which
  - given a list of students in a class
  - return a pointer to the major of the class
  - the major is the student with highest average

Linked List Exercises

1. Linked List of Contacts
2. Linked List of Integers
3. Linked List of Students
4. Linked List of Bank Accounts
5. Polynomials: Linked List of Terms
6. Doubly Linked List
7. AMSTRAMGRAM: Circular Linked List

Exercise: transfer

```
typedef struct node{
     char name[20];
     int id;
     double balance;
     struct node* next;
}node;
```

- write a function transfer which
  - given a list of bank accounts, and
  - two account number, and
  - an amount
  - transfers the given amount fro the balance of the sender (id=from) to the receiver (id=to)

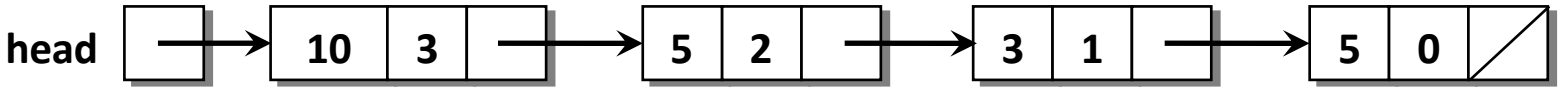
Linked List Exercises

1. Linked List of Contacts
2. Linked List of Integers
3. Linked List of Students
4. Linked List of Bank Accounts
5. Polynomials: Linked List of Terms
6. Doubly Linked List
7. AMSTRAMGRAM: Circular Linked List

Polynomial

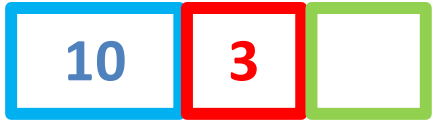
- we use linked lists to represent polynomials
- each node in the list corresponds to a term $\mathbf{c}x^{e}$, with its coefficient c and its exponent e

$$\mathbf{10}x^{3} + \mathbf{5}x^{2} + \mathbf{3}x + \mathbf{5}$$



Exercise: term

- define the data type **term** containing the following fields
  - **coef**
  - **exp**
  - **next**



Exercise: PushTerm

- write the function Push()
  - that creates and adds a new term to the head of a given list, term data are given as arguments
  - this is the classic void Push we studied in the course that takes double pointer (headRef)
- write PushTest()
  - construct the polynomial $10x^3 + 5x^2 + 3x + 5$
  - hint : put all the coefs in an array and all the corresponding exps in another array both of size 4, then loop and Push one node at a time

Exercise: printPolynomial

- write the **recursive** function **printPolynomial()**
  - that prints a given polynomial on the screen
  - example : $[10x^3 + 5x^2 + 3x + 5]$
- write **printPolynomialTest()**

Exercise: addPolynomials

- write the function **addPolynomials()**
  - that sums 2 given polynomials and returns the result
  - the result will be a 3rd polynomial allocated in heap
- write the function **addPolynomialsTest()**
  - create 2 polynomials (recall PushTest)
  - print them
  - add them
  - print the resulting polynomial

$$[10x^3 + 5x^2 + 3x + 5] + [3x^3 - 7x^2 + 3x - 6] = [13x^3 - 2x^2 + 6x - 1]$$

Exercise: addPolynomials2

- For advanced students
- Repeat the lab taking into account terms having zero coefficients $0x^6$
- Zero coefficient terms must not be present (allocated)

$$10x^7 + 5x^4 + 3x^2 + 5$$

- Rethink the **addPolynomials2()**
  - the sum of 2 terms of nonzero coefficients may lead to zero; $5x^2 - 5x^2 = 0$
  - there may be nonzero terms in first polynomial and zero terms in the second and vice-versa;

$[10x^7 + 5x^4 + 3x^2 + 5\ ] + [3x^8 + 3x^4 + 3x - 6] = [3x^8 + 10x^7 + 8x^4 + 3x^2 + 3x - 1]$

Linked List Exercises

1. Linked List of Contacts
2. Linked List of Integers
3. Linked List of Students
4. Linked List of Bank Accounts
5. Polynomials: Linked List of Terms
6. Doubly Linked List
7. AMSTRAMGRAM: Circular Linked List

Doubly Linked Lists

- Consider the following data type:

```
typedef struct node{
    int d;
    struct node *next, *prev;
} node;
```

Exercise: reverse

- Write a function "reverse" which reverses the order of the nodes in a given doubly linked list of integers.
- For example,
  - the list
    - [3⇌9 ⇌5⇌1]
  - becomes after the call to reverse
    - [1⇌ 5⇌ 9⇌ 3]

Exercise: merge

- Write a function "merge" which
  - merges two sorted doubly linked lists of integers into one.
  - The merged list must remain sorted.
  - No Node allocation or deletion is allowed.
  - Only one iteration is allowed for each list.
  - The function must return the head of the resulting merged list.

Exercise: clean

- Write the function "clean" which,
  - given a doubly linked list of integers, not sorted,
  - finds and deletes all the duplicates from the list.
- For example,
  - the list
    - [1⇌ 5 ⇌ 9⇌ 5⇌ 1⇌ 3⇌ 3⇌ 1⇌ 1]
  - becomes after the call to clean
    - [1⇌ 5⇌ 9⇌ 3].

Linked List Exercises

1. Linked List of Contacts
2. Linked List of Integers
3. Linked List of Students
4. Linked List of Bank Accounts
5. Polynomials: Linked List of Terms
6. Doubly Linked List
7. AMSTRAMGRAM: Circular Linked List

AM-STRAM-GRAM

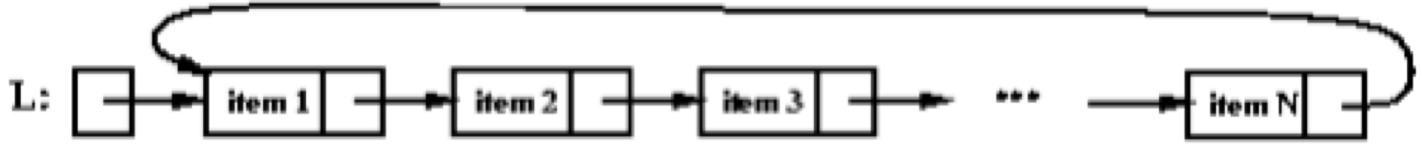
- To play the AM-STRAM-GRAM game,
  - N children form a circle,
  - choose a child to be the first,
  - start counting from this child till the k^{th} child who leaves the circle,
  - then we continue counting from current child till the next k^{th} child who will leave the circle ,
  - and so on until no child is left in the circle.

Exercise: struct child

- We suppose that a child is represented by an integer number
- Define the adequate data structure

list of children

- We represent the circle of children by a circular list



  - the last node points to the first one instead of pointing to NULL
- so to iterate down the list
  - the condition while(current!=NULL) is not valide anymore!!!
  - what is the new condition??

Exercise: play

- Write a function that displays the set of children in the order of their removal from the circle.
- Each time a child must leave the circle,
  - its node must be freed and
  - the linking must be maintained
    - the node before must points to the node after
    - unless we remove the last node in list
    - pay attention when you remove the first node (head value changes)
- The function inputs are
  - the list
  - the number k
- Example: for k = 3 and the following list :

Example: for k = 3 and the following list

1	2	3	4	5	6

Exit of 3

1	2	**4**	5	6

Exit of 6

1	2	4	5

Exit of 4

1	2	**5**

Exit of 2

1	**5**

Exit of 5

1

Exit of 1

We mark the current element in **bold**. Thus the result is: 3, 6, 4, 2, 5, 1.